MORUS-PRNG: a Hardware Accelerator Based on the MORUS Cipher and the IXIAM Framework

Alessio Medaglini, Mirco Mannino, Biagio Peccerillo, Sandro Bartolini

Email: alessio.medaglini@unisi.it

Department of Information Engineering and Mathematics, University of Siena, Italy

8th International Conference on Advanced Engineering Computing and Applications in Sciences

Session: Hardware Accelerators and Accelerated Programming

29th September, 2024

My Resume

Software engineer and **research fellow** in the Department of Information Engineering and Mathematical Sciences at the **University of Siena**.

Research topics:

- High performance computing
- Embedded and cyber-physical systems
- Software resiliency
- Cybersecurity

Ph.D. in Information Engineering from the University of Siena with a thesis titled "Object Detection and Tracking for Multi-Sensor Autonomous Driving Systems".

Work experience as **Software Engineer at Thales-GTS Italy**, developing an autonomous driving and collision avoidance system for the urban railway in Florence.

PRNG Introduction

The generation of random numbers is needed for many important applications, from numerical analysis to cryptography and security.

☹ **Bad news:** truly random numbers are difficult to generate.

☺ **Good news:** for most applications, a number sequence generated deterministically that looks random *enough* is sufficient.

To generate such a pseudo-random number sequence a hardware or software module implementing the generation algorithm is used. This module is called **Pseudo-Random Number Generator (PRNG)**.

MORUS: a NIST Cipher

PRNG implementation usually based on ciphers, which must present randomness properties. The **MORUS** cipher is one of the finalist in the CAESAR competition from NIST:

- MORUS cipher family includes 3 ciphers with **different internal state and key sizes**. Each has an internal state of 5 blocks that can have 128 or 256 bits each and deal with 128- or 256-bit keys.

- MORUS can be **efficiently implemented both in software and hardware**, since it uses only AND, XOR, shift and rotation operations.

- MORUS is “**authenticated**” because the encryption phase produces also a tag that can be used to verify decryption, providing both confidentiality and integrity.

MORUS: how it works

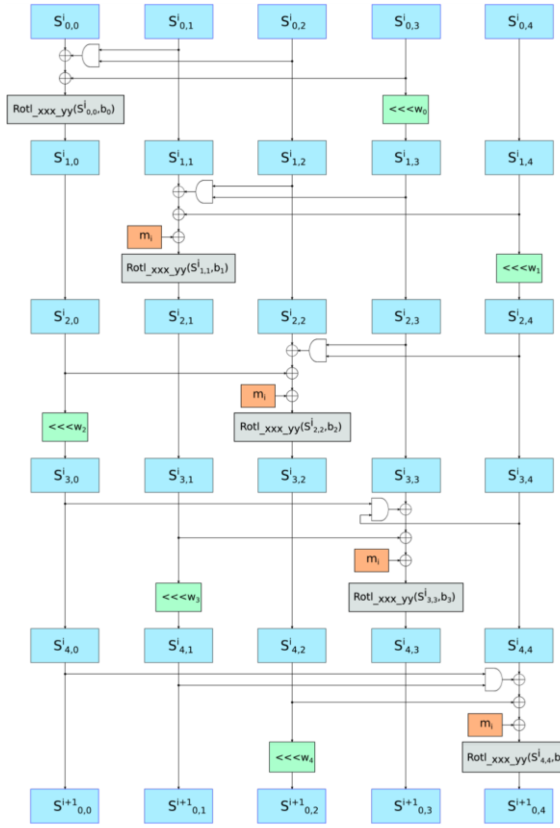
- **Initialization** reads key and Initial Value and run StateUpdate function 16 times, mixing them together into the internal state;

- **Encryption** processes one block of plain-text at a time, encrypting it with a StateUpdate call that mixes plain-text with internal state;

- **Finalization** performs a tag generation (out of our scope);

- **Decryption** is analogous to the Encryption (out of our scope).

The **StateUpdate** function consists of 5 rounds in which each block of the internal state is updated using some basic operations.

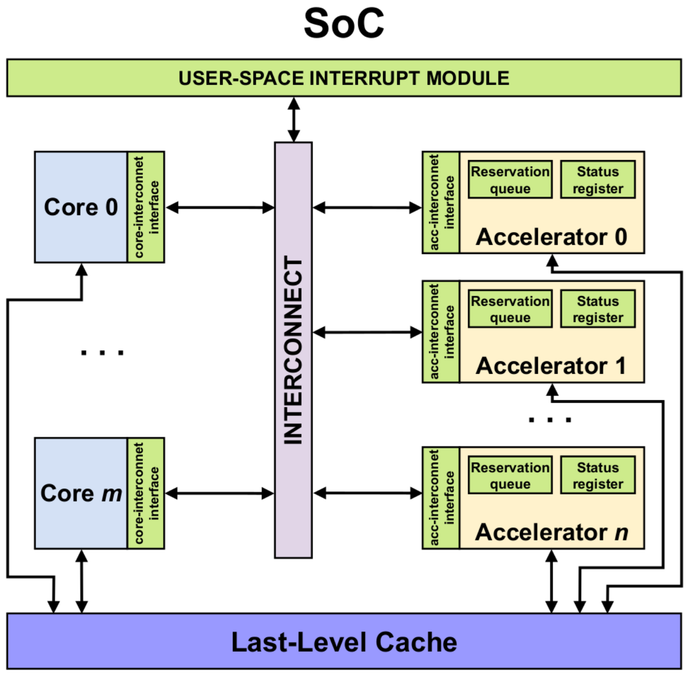

$S^i_{0,0}$ $S^i_{0,1}$ $S^i_{0,2}$ $S^i_{0,3}$ $S^i_{0,4}$
Rotl_xxx_yy($S^i_{0,0}$,b_0)
$<<<w_0$
$S^i_{1,0}$ $S^i_{1,1}$ $S^i_{1,2}$ $S^i_{1,3}$ $S^i_{1,4}$
m_i
Rotl_xxx_yy($S^i_{1,1}$,b_1)
$<<<w_1$
$S^i_{2,0}$ $S^i_{2,1}$ $S^i_{2,2}$ $S^i_{2,3}$ $S^i_{2,4}$
m_i
$<<<w_2$
Rotl_xxx_yy($S^i_{2,2}$,b_2)
$S^i_{3,0}$ $S^i_{3,1}$ $S^i_{3,2}$ $S^i_{3,3}$ $S^i_{3,4}$
m_i
$<<<w_3$
Rotl_xxx_yy($S^i_{3,3}$,b_3)
$S^i_{4,0}$ $S^i_{4,1}$ $S^i_{4,2}$ $S^i_{4,3}$ $S^i_{4,4}$
m_i
$<<<w_4$
Rotl_xxx_yy($S^i_{4,4}$,b_4)
$S^{i+1}_{0,0}$ $S^{i+1}_{0,1}$ $S^{i+1}_{0,2}$ $S^{i+1}_{0,3}$ $S^{i+1}_{0,4}$

IXIAM Framework

ISA eXtension for Integrated Accelerator Management (IXIAM) is a **hardware-software framework for SoCs** composed by:

- reservation queue and status register for each accelerator;
- core- and accelerator- interconnect interface;
- user-space interrupt module.

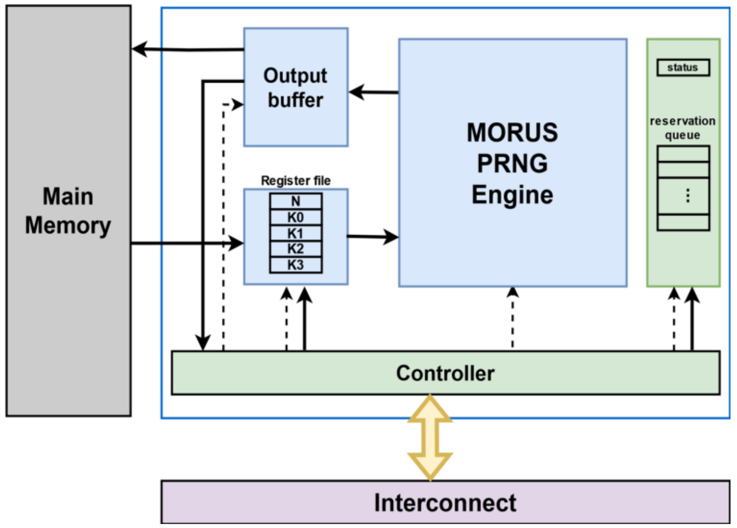
It allows controlling accelerators directly from the cores, **ensuring low communication latency** and significant performance advantages over drivers.



Our Proposal

We propose an integrated MORUS-based PRNG hardware accelerator based on the IXIAM framework:

- **MORUS PRNG Engine** for number generation;
- **Output buffer** to place array of generated numbers in;
- **Register file** to hold the amount of numbers to be generated and four key registers;
- **Controller** responsible for managing the IXIAM packets;
- **IXIAM HW infrastructure** with a status register and a reservation queue.



Our Proposal: Initialize operation

The Initialize operation is responsible for triggering the initialization phase in the MORUS cipher:

1. **Load values into the initial state:** the user is responsible for writing the four words composing the 128-bit key, used as seed for the generator;

2. **Trigger initialization:** when the accelerator receives the EXEC packet from IXIAM, the MORUS PRNG Engine loads key and initial value and execute the MORUS initialize procedure;

3. **MORUS initialized:** the engine is ready to perform encryptions, necessary to generate pseudo-random numbers. An internal 128-bit counter, inaccessible from outside, is set to 0.

Our Proposal: Generate operation

The Generate operation performs the encryption phase in the underlying MORUS cipher:

1. **Reading of register N:** register written by the user and interpreted by the engine as the amount of 32-bit pseudo-random numbers to generate;

2. **Pseudo-random number generation:** performed by encrypting the content of the internal counter. At each step, a 128-bit block of ciphertext, which can be interpreted as four 32-bit numbers, is generated this way.

3. **Output buffer:** the generated numbers are written into the output buffer starting from address 0.

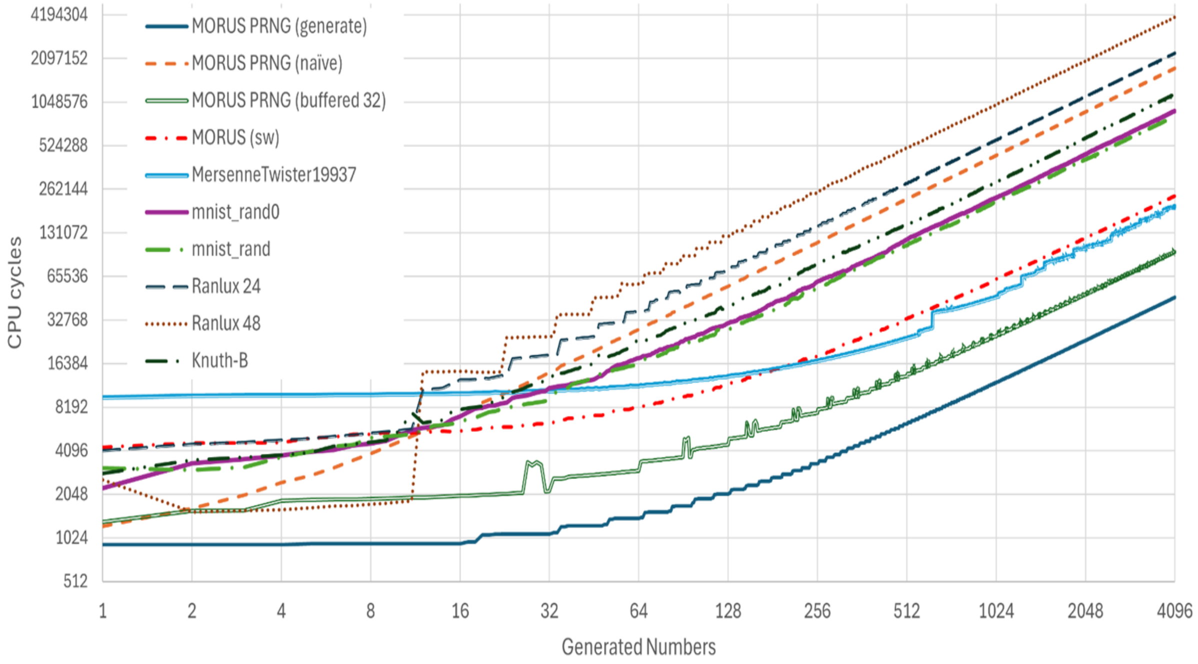
Evaluation

Implementation based on the **gem5 architecture simulator**, with a Generator class in C++ wrapping communication to the accelerator. It executes Initialize when constructed and exposes two methods:

- **operator():** outputs a single generated number, we design two variants of it:
    - **naïve:** generates 1 number on the accelerator and returns it to the caller;
    - **buffered:** generates few numbers on accelerator and returns a buffered number at each call.
- **Generate:** fills an array with N generated numbers;

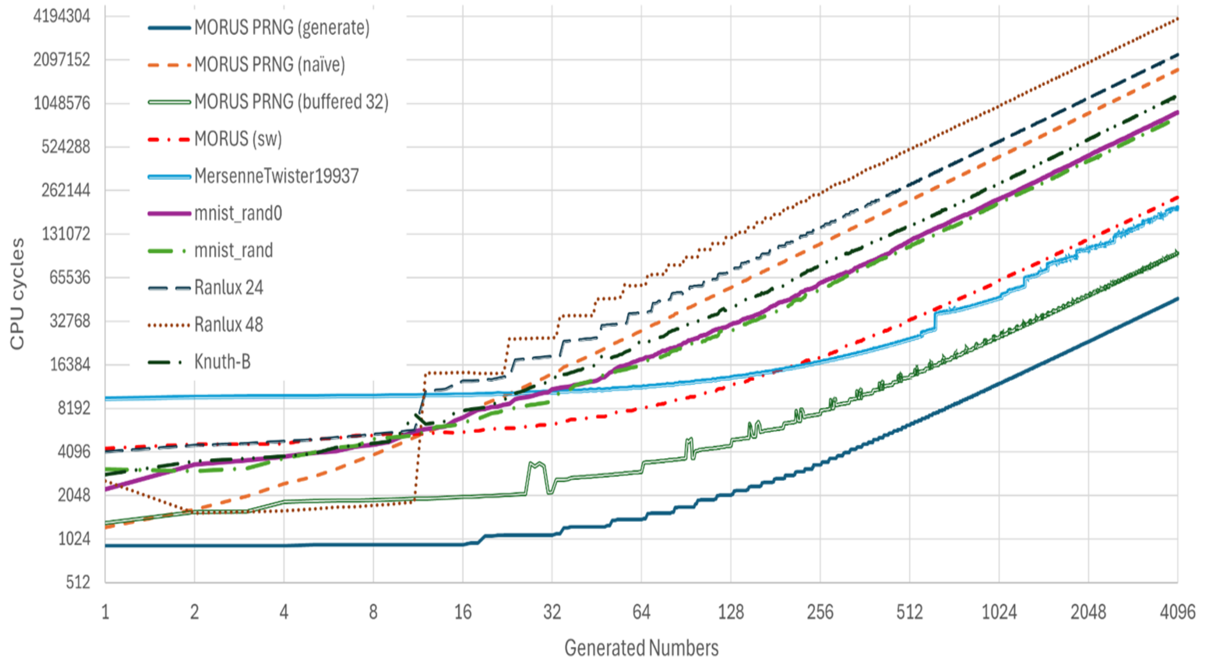
To evaluate our solution, we compare it with other PRNGs defined in the random C++ header. Performance is measured in CPU cycles (lower is better).

Results



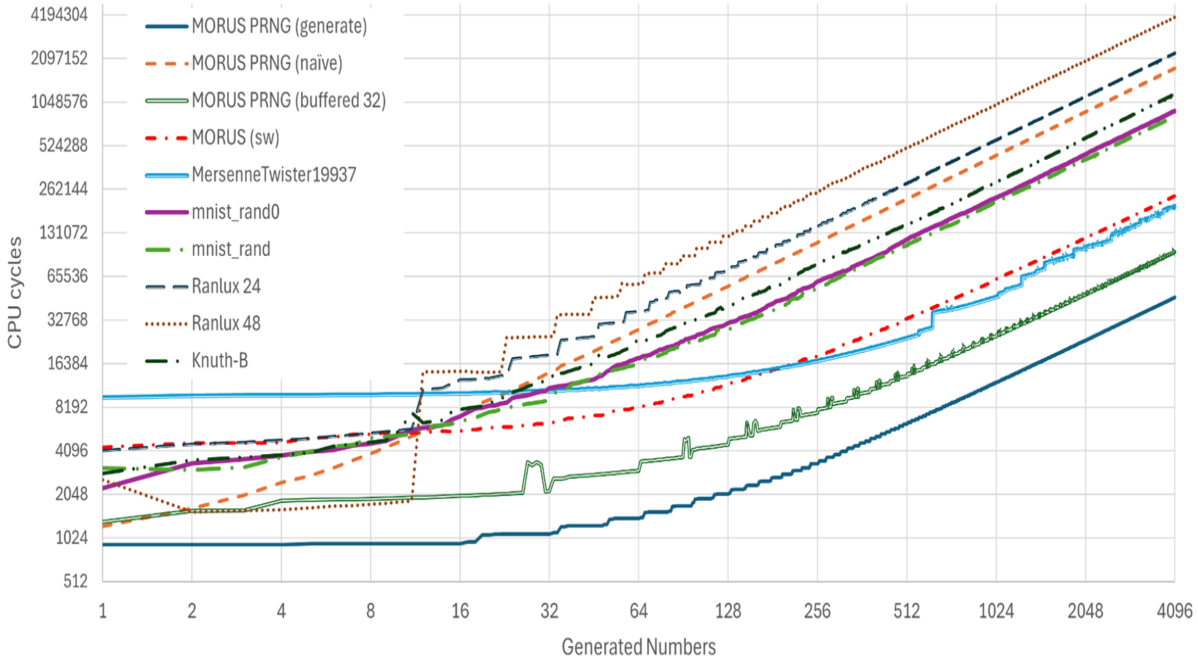
- **Generate** outperforms other PRNGs. Performance gap rising with size up to 4.26x.

Results



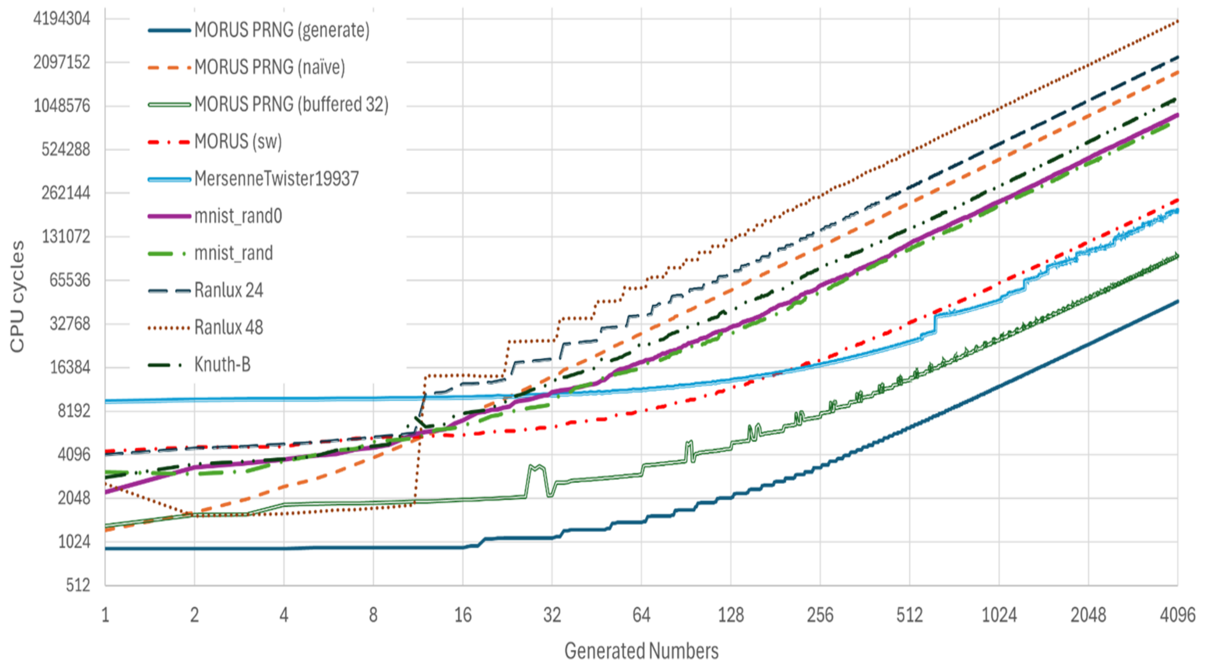
- **Generate** outperforms other PRNGs. Performance gap rising with size up to 4.26x.
- **Naïve** pays latency in communication between CPU and accelerator.

Results



- **Generate** outperforms other PRNGs. Performance gap rising with size up to 4.26x.
- **Naïve** pays latency in communication between CPU and accelerator.
- **Buffered** outperforms other PRNGs for size greater than 11. Speedup up to 2.07×.

Results



- **Generate** outperforms other PRNGs. Performance gap rising with size up to 4.26x.
- **Naïve** pays latency in communication between CPU and accelerator.
- **Buffered** outperforms other PRNGs for size greater than 11. Speedup up to 2.07×.
- **MORUS-sw** is faster than all the other PRNGs for sizes between 12 and 196.

Conclusion

- In this work, an integrated accelerator for pseudo-random number generation based on the **MORUS cipher** was proposed.
- We designed it to communicate with the CPU through the **IXIAM framework**, which allows users to control it directly with CPU instructions ensuring a lower latency.
- We evaluated it in a simulated environment in the **gem5 architectural simulator**, comparing its performance against PRNGs included in the C ++ standard library.
- We showed that it is able to **outperform** them.

Thank you for your attention

Any questions?

Alessio Medaglini

Email

alessio.medaglini@unisi.it